

First Hit   Fwd Refs  
End of Result Set

☐ **Generate Collection**   **Print**

L7: Entry 1 of 1

File: USPT

Jul 24, 2001

DOCUMENT-IDENTIFIER: US 6266666 B1

TITLE: Component transaction server for developing and deploying transaction-intensive business applications

Abstract Text (1):

A Component Transaction Server (CTS) is described, which provides a framework for deploying the middle-tier logic of distributed component-based applications. The CTS simplifies the creation and administration of Internet applications that service thousands of simultaneous clients. The CTS components, which execute on the middle-tier between end-user client applications and remote databases, provide efficient management of client sessions, security, threads, third-tier database connections, and transaction flow, without requiring specialized knowledge on the part of the component developer. The system's scalability and platform independence allows one to develop application on inexpensive uniprocessor machines, then deploy the application on an enterprise-grade multiprocessor server. In its Result Set module, the CTS provides tabular result sets, thus making the environment very desirable for business applications. In most component-based systems, a component interface returns an object. CTS components can return either an object or a collection of objects called a "result set." The format of a result set is based on the standard ODBC result set, and it is roughly equivalent to a database cursor. Because they return a result set, CTS components are much simpler and more efficient to work with. In this fashion, graphic user interface (GUI) development with CTS is nearly identical to traditional two-tier systems.

Brief Summary Text (9):

The three-tier design has many advantages over traditional two-tier or single-tier designs. For example, the added modularity makes it easier to modify or replace one tier without affecting the other tiers. Separating the application functions from the database functions makes it easier to implement load balancing. Thus, by partitioning applications cleanly into presentation, application logic, and data sections, the result will be enhanced scalability, reusability, security, and manageability.

Brief Summary Text (10):

In a typical client/server environment, the client knows about the database directly and can submit a database query for retrieving a result set which is generally returned as a tabular data set. In a three-tier environment, particularly a component-based one, the client never communicates directly with the database. Instead, the client typically communicates through one or more components. Components themselves are defined using one or more interfaces, where each interface is a collection of method. In general, components return information via output parameters. In the conventional, standard client/server development model, in contrast, information is often returned from databases in the form of tabular result sets, via a database interface such as Open Database Connectivity (ODBC). A typical three-tier environment would, for example, include a middle tier comprising business objects implementing business rules (logic) for a particular organization. The business objects, not the client, communicates with the database. Nevertheless, there is still the desire to have client/server features, such as scrollable

results and data input fields.

Brief Summary Text (11):

Since existing tools were created for the client/server model, they are better suited or adapted to receiving table-based or tabular "result sets" for retrieving complex data. Also, tabular data tends to be a more efficient way to retrieve data from a relational database, such as from Sybase.RTM. SQL Server. At the same time, however, components provide distinct advantages. In particular, they provide a well-designed model and interface, so that reusable code is achievable with minimal effort.

Brief Summary Text (13):

What is needed is a solution in which provides a simplified database connectivity interface for communicating with components, including Java and ActiveX components, as well as conventional C components, such that the interfaces of the components can be called for returning a tabular result set to the client. The present invention fulfills this and other needs.

Brief Summary Text (17):

The CTS kernel includes a Session Management module, a Security module, a Result Set module, a Thread Polling module, a Transaction Management module, and a Connection Pooling module. Components execute within the CTS multithreaded kernel which provides automatic optimization of available system resources. By pooling and reusing resources such as threads, sessions, and database connections, the CTS eliminates significant system overhead. Here, CTS components execute on native threads within the CTS kernel. The CTS maintains a pool of threads and allocates them to components as needed. Components can be configured as single-threaded or multithreaded. If multithreaded, a thread can be allocated for the life of the component instance, or, for better scalability, a new thread can be allocated on each method invocation.

Brief Summary Text (18):

In the Result Set module, the CTS provides tabular result sets, thus making the environment very desirable for business applications. In most component-based systems, a component interface returns an object. CTS components can return either an object or a collection of objects called a "result set." The format of a result set is based on the standard ODBC result set, and it is roughly equivalent to a database cursor. Because they return a result set, CTS components are much simpler and more efficient to work with. For instance, result sets can be bound directly to data-aware controls such as a PowerBuilder.RTM. DataWindow, a Sybase.RTM. PowerJ Grid control, or any data-aware control that can be bound to an ODBC result set. In this fashion, GUI development with CTS is nearly identical to traditional two-tier systems. The CTS automatically manages partial refreshes and updates of the result set.

Brief Summary Text (19):

The CTS is middleware, an application server. Components installed on the CTS become usable across whatever network the CTS is connected to (which can include the Internet). In accordance with the present invention, a simplified database connectivity interface is provided for communicating with components, including JavaBeans, Java classes, ActiveX controls, and even DLLs (Dynamic Link Libraries). The interfaces of the components can be called for returning a tabular result set to the client. In a corresponding manner, a component (i.e., collection of methods) is provided at the client such that any one of its methods is marked for returning tabular results. When the system generates a stub for Java or a proxy for ActiveX, the system makes tabular results available through standard interfaces. In the instance of a Java component, for example, the system makes the results available through the JDBC (Java Database Connectivity) interface. Here, a client can invoke a method on a component and request the results as a tabular data set. For a Java component, the client can request a JDBC result handle (i.e., database cursor). In

a visual development environment, such a handle could then be bound to a visual control or component, provide providing both component-based development and tabular data.

Drawing Description Text (5):

FIGS. 3-6 are bit map screen shots illustrating a visual development environment (e.g., PowerBuilder.RTM.) where a client can invoke a method on a component and request the results as a tabular data set, using the system of the present invention.

Detailed Description Text (6):

Illustrated in FIG. 1B, a computer software system 150 is provided for directing the operation of the computer system 100. Software system 150, which is stored in system memory 102 and on mass storage or disk memory 107, includes a kernel or operating system (OS) 140 and a windows-based GUI (graphical user interface) shell 145. One or more application programs, such as application software programs 155, may be "loaded" (i.e., transferred from storage 107 into memory 102) for execution by the system 100. The system also includes a user interface 160 for receiving user commands and data as input and displaying result data as output.

Detailed Description Text (17):

As shown in FIG. 2, the CTS kernel includes a Session Management module 222, a Security module 223, a Result Set module 224, a Thread Polling module 225, a Transaction Management module 226, and a Connection Pooling module 227. Components execute within the CTS multithreaded kernel which provides automatic optimization of available system resources. By pooling and reusing resources such as threads, sessions, and database connections, the CTS eliminates significant system overhead. Here, CTS components execute on native threads within the CTS kernel. The CTS maintains a pool of threads and allocates them to components as needed. Components can be configured as single-threaded or multithreaded. If multithreaded, a thread can be allocated for the life of the component instance, or, for better scalability, a new thread can be allocated on each method invocation.

Detailed Description Text (22):

In the Result Set module 224, the CTS 221 provides tabular result sets, thus making the environment very desirable for business applications. In most component-based systems, a component interface returns an object. CTS components can return either an object or a collection of objects called a "result set." The format of a result set is based on the standard ODBC result set, and it is roughly equivalent to a database cursor. Because they return a result set, CTS components are much simpler and more efficient to work with. For instance, result sets can be bound directly to data-aware controls such as a PowerBuilder.TM. DataWindow, a Sybase.RTM. PowerJ Grid control, or any data-aware control that can be bound to an ODBC result set. In this fashion, GUI development with CTS is nearly identical to traditional two-tier systems. The CTS 221 automatically manages partial refreshes and updates of the result set.

Detailed Description Text (26):

The CTS is middleware, an application server. Components installed on the CTS become usable across whatever network the CTS is connected to (which can include the Internet). In accordance with the present invention, a simplified database connectivity interface is provided for communicating with components, including JavaBeans, Java classes, ActiveX controls, and even DLLs (Dynamic Link Libraries). The interfaces of the components can be called for returning a tabular result set to the client. In a corresponding manner, a component (i.e., collection of methods) is provided at the client such that any one of its methods is marked for returning tabular results. When the system generates a stub for Java or a proxy for ActiveX, the system makes tabular results available through standard interfaces. In the instance of a Java component, for example, the system makes the results available through the JDBC (Java Database Connectivity) interface. Here, a client can invoke

a method on a component and request the results as a tabular data set. For a Java component, the client can request a JDBC-result-handle (i.e., database cursor). In a visual development environment, such a handle could then be bound to a visual control or component, provide providing both component-based development and tabular data.

Detailed Description Text (31):

Proceeding with the example, the user will create a new data window, such as a stored procedure data window, for illustrating that a method on a component can be made to look like a stored procedure (i.e., returning a tabular result set). At this point, the PowerBuilder environment sends a query requesting a list of stored procedures. The component transaction server, in response, enumerates what components are on the transaction server and what methods are on those components. Thereafter, the component transaction server returns a list of the enumerated items, in effect making them appear as storage procedures to PowerBuilder, as shown by dialog 310.

Detailed Description Text (32):

Consider as an example, for instance, a component named "book store" with a method entitled "get books by title." Using a conventional approach, there is no way of knowing what results the method will return if one were to simply use querying technique. Therefore, conventionally, one is required to know beforehand what results are returned by the method. Using the approach of the present invention, however, this is not required. As shown in FIG. 4, the user simply proceeds as follows. First, the user enters expected data types, such as a text type for title and a real (floating point) type for price, as shown in dialog 410, and then specifies retrieve arguments, as shown in dialog 510 in FIG. 5. Then, after specifying argument information and indicating a start and a finish, the information can be executed against the component residing on the middle tier, not against the database. The data is returned to the client as a tabular data set, despite the fact that the method itself is simply one which returns books by title. This is shown at 610 in FIG. 6.

Detailed Description Text (35):

An important aspect of the design is the ability of a component to return a result back to the client. From the perspective of a component writer, the present invention provides an application programming interface (API) for "pushing" a data set to clients. Interface calls are provided for describing every column, binding variables to those columns, initializing those variables, and then sending the data. In the currently-preferred embodiment, the interface is defined for Java, ActiveX, and C, thus allowing all component models supported by the system a means to send tabular result sets.

Detailed Description Text (37):

On the client side, the user is able to generate a component graphically (e.g., using PowerBuilder or other visual development environment). For a Java component, the system of the present invention generates a stub. The stub, which resembles a Java class, include employs JDBC for sending requests to the component transaction server for receiving tabular result sets back.

Detailed Description Text (42):

Component metadata is also used to support PowerBuilder.TM. stored procedure DataWindow.TM.. Powerbuilder queries the CTS as though it were a database, asking it what stored procedures are available. The CTS responds by delivering a tabular result set containing the names of all the available component methods. This information is acquired by iterating through the metadata hash table that was created at startup by one of Metadata's methods, Metadata::loadMetadata( ).

Detailed Description Text (52):

The format of the command name is <package>.<component>.<method>. Then,

runJaguarStoredProc( ) looks up the method in the Metadata cache and, if it is found, creates the component, calls the method on the component, returns the results to the client, and then destroys the component. The basic approach can be summarized as follows, again in pseudo-code.

Detailed Description Text (59):

Although the mechanisms for accessing the procedure name and parameters are quite different, the end result is identical: a client is requesting an invocation of a component method. This similar behavior is encapsulated in the abstract Marshaller class. There are two concrete implementations of this class, TDSMarshaller and LangMarshaller. TDSMarshaller is a Marshaller that handles RPC requests, including Sybase(K TDS protocol.

Detailed Description Text (65):

Once the "method def" is set up, Command::runJaguarStoredProc then invokes the method. This results in a component skeleton or dynamic dispatcher being called. This skeleton calls the Marshaller method getAParam( ) whenever it needs a parameter. It receives a ParamData object which contains not only the parameter data but information about the parameter such as its maximum length and its datatype, as follows.

Detailed Description Text (69):

3. Sending Results

Detailed Description Text (70):

A particular advantage of the MASP interface is that it truly does look like a database stored procedure. This includes the ability for a CTS method to send back tabular result sets as if these results came from a database table.

Detailed Description Text (71):

The CTS provides APIs in C, ActiveX and Java, that allow a component writer to describe, populate, and deliver tabular results to the client. These APIs, for the most part, map to an open interface, such as APIs in Sybase.RTM. Open Server.TM.. There is minimal new code in Jaguar to make this work--most of it is mapping between the component's language and/or component model and Open Server.

CLAIMS:

1. In a system comprising a computer network having a database server and a client, a method for providing the client with component-based access to tabular data from the database server, the method comprising:

providing a component-based transaction server, said transaction server in communication with both the client and the database server;

registering the particular component with the transaction server so that the particular component is accessible across the network to the client, said particular component comprising a plurality of methods; and

providing the client with component-based access to tabular data from the database server by:

(i) providing an interface allowing the client to request a tabular data set by invoking a particular method of the component; and

(ii) upon receiving such a request from the client to retrieve data from the database server, processing said request by creating at the transaction server a result set satisfying said request and transmitting said result set to the client as a tabular data set.

8. The method of claim 1, wherein step (ii) includes transmitting from the database server to the transaction server a tabular result set satisfying said request.

9. The method of claim 8, wherein said tabular result set is transmitted from the database server to the transaction server as a result of a database query transmitted from the transaction server to the database server.

16. A system for providing a client with component-based access to tabular data from a database server, the system comprising:

a network connected to at least one database server and at least one client;

a transaction server connected to the network and in communication with both a particular client and a particular database server, said transaction server for providing the particular client with component-based access to tabular data from the particular database server; and

a connectivity interface allowing the particular client to request a tabular data set by invoking a particular method of a component registered with the transaction server;

wherein upon receiving such a request from the particular client to retrieve data from the particular database server, the transaction server processes said request by retrieving from the particular database server a result set satisfying said request and transmitting said result set to the client as a tabular data set.

23. The system of claim 16, wherein transaction server processes said request, at least in part, by retrieving into a local buffer of the transaction server a tabular result set satisfying said request.

24. The system of claim 23, wherein said tabular result set is retrieved as a result of a database query transmitted from the transaction server to the database server.